# 2DX3: Microprocessor Systems Project

Final Written Report

Zavi Jawaid (jawaidz, 400368132)

Department of Electrical and Computer Engineering, McMaster University

Dr. Yaser Haddara, Dr. Thomas Doyle, Dr. Shahrukh Athar

April 17, 2023

## Device Overview

***Features***

- Creates a 3D reconstruction of a space by scanning it
- Employs the use of the MSP432E401Y microcontroller and Keil UVision Development Tool in C Programming Language
- Runs at a bus speed of 24 MHz
- Operates using the VL53LIX Time of Flight Sensor and the MOT-28BYJ48 Stepper Motor and Driver
- Has a singular, external push-button to start and stop the rotatory device
- Has operating voltages of 3.3 V and 5 V
- Using PySerial and Open3D libraries, is able to generate high quality 3D recreations
- Functions using $I^2C$ and UART (11520 bps) in order to send and receive data from the microcontroller to the PC
- Uses two onboard LED status trackers, D3 for measurement status and D4 for ToF status
- Total cost of approximately 200$ CAD

***General Description***

This project attempts to create a portable and reasonably priced replacement for commercial Light Detection and Ranging (LIDAR) equipment, by utilising a time-of-flight sensor and a rotational mechanism to collect spatial data in a single vertical geometric plane. The gathered data is saved in onboard memory and relayed to a computer via a connected serial interface, where it is reconfigured to represent a three-dimensional space.

The project does this by rotating a VL53L1X time-of-flight sensor exactly by 11.25 degrees after each data acquisition for 32 times in a set to gather 32 points per 360 degrees. The sensor is coupled to a MOT-28BYJ48 stepper motor. The sensor and motor are both controlled by the MSP432E401Y microcontroller, which is powered by a micro-USB connection attached to the PC. It supports UART and $I^2C$ serial connection techniques and contains 1024 kilobytes of flash memory and 256 kilobytes of static random-access memory. It was developed in C. The system also has a single push button that turns the stepper motor and the ToF sensor in addition to starting and stopping the data collecting operation.

To give the time-of-flight sensor a clear view of the 360-degree environment, the sensor is linked to the system using an extension portion. The measurement is first saved in local memory before being transmitted to the PC using the UART serial communication protocol. The sensor connects with the microcontroller using the I2C serial communication protocol. Finally, Python code running on the computer parses and visualises the gathered data.

In summary, using a time-of-flight sensor and a rotational mechanism to collect spatial measurements in a single vertical geometric plane, this project successfully builds a cheap and portable substitute for commercial LIDAR equipment. The sensor and the motor are controlled by the microcontroller MSP432E401Y, and the data that is collected is sent to the PC via serial communication protocols so that Python code can visualise it.

***Block Diagram (Data Flow Graph)***

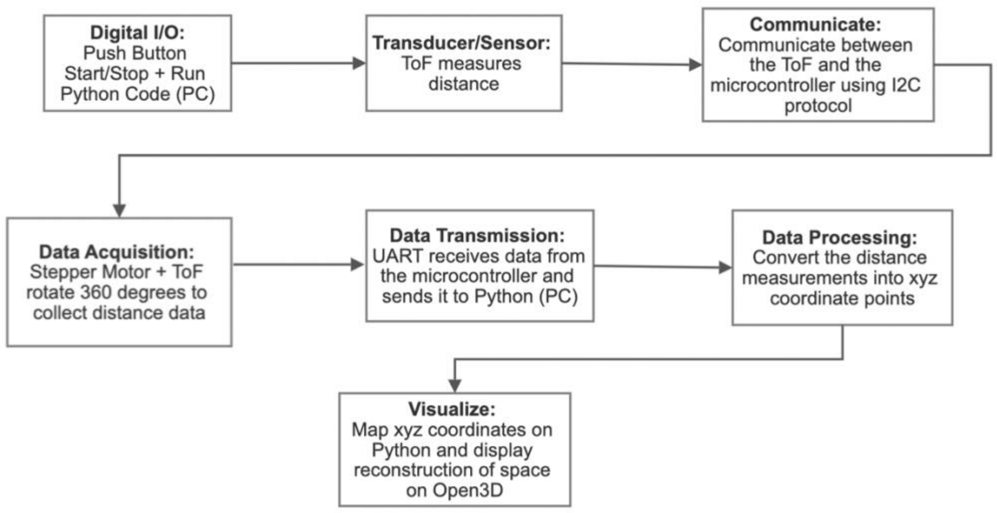
**Digital I/O:** Push Button Start/Stop + Run Python Code (PC)
**Transducer/Sensor:** ToF measures distance
**Communicate:** Communicate between the ToF and the microcontroller using I2C protocol
**Data Acquisition:** Stepper Motor + ToF rotate 360 degrees to collect distance data
**Data Transmission:** UART receives data from the microcontroller and sends it to Python (PC)
**Data Processing:** Convert the distance measurements into xyz coordinate points
**Visualize:** Map xyz coordinates on Python and display reconstruction of space on Open3D

*Figure 1 – Block Diagram of Data Flow*

## Device Characteristics Table

| *Device Characteristics* | *Values* |
|---|---|
| Bus Speed | 24 MHz |
| Digital I/O Distance Measurement Status | PF4 (LED D3) |
| Digital I/O ToF Status | PF0 (LED D4) |
| Stepper Motor Driver Pins | IN1-IN4, + and - (on Stepper Motor)<br>PL0-PL3 (on microcontroller) |
| Time of Flight Sensor Pins | PB2 (I2C SCL)<br>PB3 (I2C SDA) |
| External Push Button Output Pin | PH0 |
| Operating Voltages | 3.3 V (ToF)<br>5 V (External Push Button, Stepper Motor) |
| Baud Rate | 115200 bps |
| Python Software | v.3.6 – 3.10 (Open3D not compatible with v.3.11) |
| Python Libraries | PySerial, Open3D |

## Detailed Description

***Distance Measurement***

The VL53L1X time-of-flight sensor is employed by the system. There are two surfaces on the time-of-fight sensor. One of them sends a light beam in the direction of the target, and the other one collects the light that is reflected. The time it takes for light to travel to the target and return can be used to determine how far away the target is from the sensor because the speed of light is a known constant.

There are 4 connections on the ToF. On the microcontroller, GND and VIN are linked to ground and 3.3 volts, respectively. Also, SDA and SCL pins are attached to PB3 and PB2 on the microcontroller, respectively, to provide an I2C serial connection in order to communicate with the sensor. The VL53L1X API library is used to configure and operate the sensor. The given SensorInit function is used to configure the sensor, and the StartRanging method is used to measure the distance. The GetDistance function is then used to get the measured distance.

Then, the sensor is accurately rotated clockwise by the stepper motor between each measurement. The IN1, IN2, IN3, and IN4 pins are connected to the microcontroller's PL0, PL1, PL2, and PL3 ports, respectively. Additionally, the microcontroller's - and + pins are connected to ground and 5 volts, respectively. The four electromagnets inside the stepper motor are switched around in polarity to produce motion. The shaft between the magnets can revolve in both rotations by switching the polarities in the proper order.

The process used to switch polarities, known as full step, where two neighbouring magnets are powered at all times. The algorithm waits 10 milliseconds in between each state change to give the shaft ample time to rotate. The 32 measurements per 360-degree set will be taken by the system because the resolution is set to 32. As a result, following each data capture, the stepper motor will rotate the sensor 11.25 degrees.

C is the programming language used for the microcontroller. According to the flowchart of the microcontroller code in Figure 6, the main code operates as a finite state machine in an infinite loop. Phase-Locked Loop (PLL) is set for the bus speed, SysTick is set for waiting, onboard LEDs are set for visual output, I2C is configured for sensor connection, UART is set for PC communication, and GPIO ports are set for button inputs. All necessary functionality is then initialised by the code. The time-of-flight sensor is then initialised and set up. When the sensor is prepared, it communicates the word "start" over UART and starts an endless loop.

The code checks to see if the button linked to the PH0 is pressed before entering the infinite loop. It will start the data collecting process once the button is pressed. All of the onboard LED's flash to signal the commencement, and a 50 millisecond delay is then allowed. The time-of-flight sensor is then instructed to begin calculating the range. The program checks every 5 milliseconds for the availability of sensor data and will flash the onboard LED attached to the PF0 port. When the information is ready, it uses I2C to temporarily save the distance data in onboard memory after retrieving it from the sensor. The sensor is subsequently rotated clockwise by 11.25 degrees by the stepper motor. Following the rotation, the distance information is sent to the PC via UART, as described in the following section. When the transmission is finished, it flashes the onboard LED attached to the PF4 port, before waiting 100 milliseconds. The button PH0 is then once more examined to see if it has been pressed to halt the operation. If it is, the operation is stopped, and the program loops indefinitely. Depending on the number of measurements listed in the code as resolution, each of the aforementioned steps—starting with waiting for data availability—is repeated.

***Visualization***

The time-of-flight sensor distance measurements in the XY plane are supplied to the computer via UART serial communication. The microcontroller is set up so that UART0 is preferred over the micro-USB for power and code flashing. UART is set up to use an 8-bit word length, a baud rate of 115200, no parity bits, one stop bit, and FIFOs. The interface between the microcontroller and the PC is created using a Python collector code.

Figure 7 - Flowchart of the PC Collector Code shows the logic as a flowchart. Since the time-of-flight sensor status during start-up is also sent by the VL53L1X library internally using UART, the collector must be informed when the actual distance data is about to be transmitted. When the setup code has properly completed, the microcontroller transmits the word "start," and the collector waits until it reads that word before continuing.

$$y = \frac{distance}{scanned\ sets} \times sin\left(\frac{2\pi}{scans} \times measurements\right) \quad [1]$$

$$z = \frac{distance}{scanned\ sets} \times cos\left(\frac{2\pi}{scans} \times measurements\right) \quad [2]$$

$$x = 50 \times counter \quad [3]$$

The data collecting logic begins to run as soon as the word "start" is received by the collector code over UART. The microcontroller ends each line of data it transmits with a carriage return ("r") and a line feed ("n") character. In this manner, the receiver may quickly determine when all the data for the most recent distance measurement has been transmitted. The collector initially converts each line of input into an integer before dividing each line's received distance information into its Y and Z components in accordance with formulas [1] and [2]. The value obtained from the microcontroller is the "distance". The number of times the system will move down the X axis and take a complete set of measurements depends on the "scanned sets" setting. How many measurements will be in a set is determined by the "scans" value. The "measurements" parameter also maintains track of how many measurements are currently in the set. The "counter" value in formula [3] is used to determine the X value and keeps track of the current number of sets. In relation to how far the system is pushed along the X axis, the value 50 is arbitrarily selected.

The calculated XYZ values are then appended to the file "data.txt," which the visualizer code can access. Each line in the file thus reflects a measurement of the spatial area in the format "X Y Z \n". The file is opened and closed before and after each set in order to improve the file input/output reliability.

As shown in Figure 8 - flowchart of the PC visualizer code, the visualizer code reads the XYZ data that the collector code has generated. The programme transforms the XYZ data into a point cloud object using the Open3D library. Even while it is possible to display this data directly as dots in 3D space, the final product is difficult to understand. As a result, a line set is produced by joining each point to its neighbouring points. Both internal connection lines within the set and external connecting lines between the sets are included in this set.

## Application Example

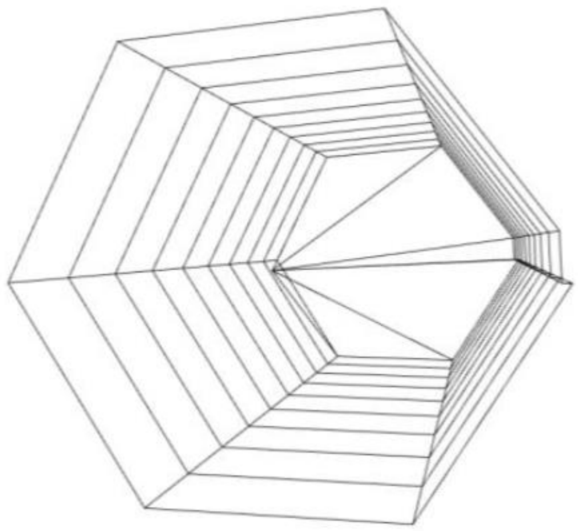

*Figure 2 – 3D Reconstruction of hallway in ITB*

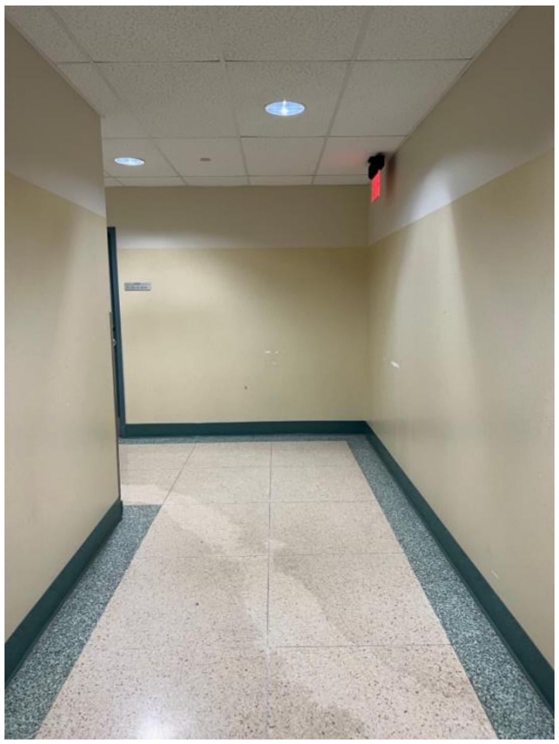

*Figure 3 – Hallway ‘C’ in ITB*

Figure 2 – A 3D reconstruction of hallway ‘C’ in ITB as located on the second floor (Figure 3). The scans were taken by taking 8 sets of scans and taking displacements of approximately 500-600 mm between each scan set. The scans were also taken by tilting the system so that ToF had the ability to take a 360 view, while facing forward.

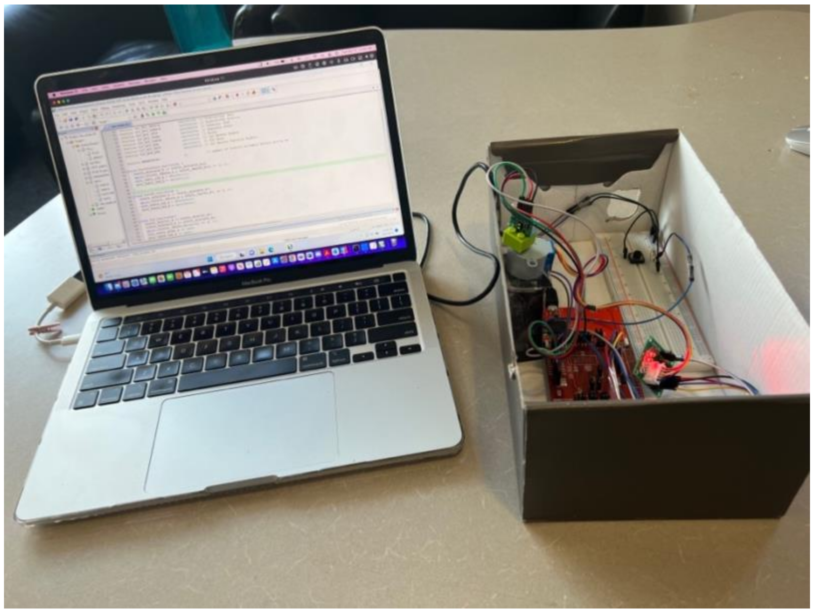
*Figure 4a – Photo of Physical Set-up*

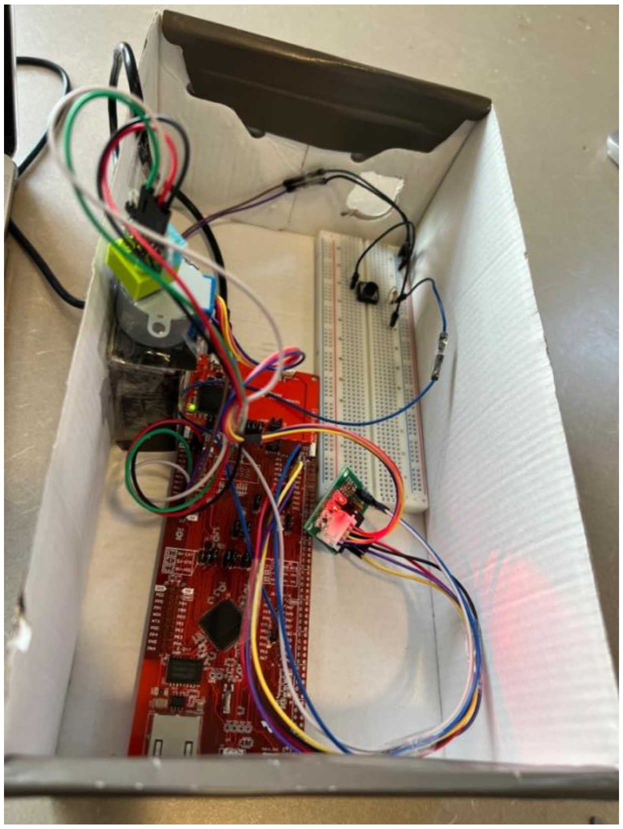
*Figure 4b – Overview of System*

## ***User Guide***

1. Set-up Python to be able run the collector and visualizer files:

    a. Install Python v.3.6 – 3.10 (PySerial not supported with v.3.11)
    b. Install Python libraries PySerial, numpy and Open3D through the terminal or command port of your PC
2. Set-up the microcontroller:
    a. Connect the microcontroller you your PC through a USB port
    b. Download the Keil code on to microcontroller
3. Find the serial port for your computer using the terminal or command port of your PC (should be in the form of "COMX" for Windows or "/dev/cu.usbmodemME4010231" for mac)
4. Set-up the physical circuit by referring to the circuit schematic in Figure 5
5. Run the CollecterDoc.py code
6. Press the reset button on the microcontroller to reset
7. Wait for all components to initialize and wait for the "start" message to appear on the terminal
8. Press the button connected to PH0 on the microcontroller, which will start the data acquisition process for the first set
    a. At this point, you should see a series of distance values being printed onto the terminal screen
9. Wait for all 32 distance scans to appear on the screen followed by a "set-x" message that indicates the system is ready to receive the next set of scans
10. Displace yourself 50-60 cm to the next scanning point
11. Repeat steps 8-10 until all scan sets have been completed
    a. The program should now have come to a halt
12. Run the VisualizerDoc.py code
13. The values have now been successfully plotted and constructed as a 3D reconstruction

## Limitations

(1) The Floating-Point Unit (FPU) offers floating-point constant instructions, more precisely 32-bit instructions for single-precision (C float) data-processing operations, as well as conversions between fixed-point and floating-point data formats. Python accepts integer distance values. By importing the math Python library, trigonometric functions were added to the Python program. Since Python allows for the casting of an integer into a float, utilising the sine and cosine functions from the math package to perform trigonometry did not result in any restrictions.

(2) The maximum quantization for each of the ToF modules can be determined using equation [4], where "V" is equivalent to 5 V and "n" is equivalent to the number of bits. Therefore, the maximum quantization error is calculated to be 19.53 mV [5], for the ToF sensor which has an 8-bit ADC. The maximum quantization error is calculated to be 1.22 mV [6] for the MSP432E401Y, which has a 12-bit ADC. Lastly, the maximum quantization error is 76.30 uV [7] for the IMU sensor which has a 16-bit ADC.

$$maximum\ quantization\ error = \frac{V}{2^n}\ [4]$$

$$maximum\ quantization\ error\ (ToF) = \frac{5}{2^8} = 19.53\ mV\ [5]$$

$$maximum\ quantization\ error\ (MCU) = \frac{5}{2^{12}} = 1.22\ mV\ [6]$$

$$maximum\ quantization\ error\ (IMU) = \frac{5}{2^{16}} = 76.30\ \mu V\ [7]$$

(3) The standard serial transmission rate used in this project, which is the highest possible rate, is 115.2 kbps. For the vast majority of microcontrollers, including MSP432E401Y, this is considered quick. When higher speeds were attempted and transmission faults were found, this maximum speed was confirmed.

(4) Data transfer is done via the I2C (UART) serial communication protocol at a speed of 100kbps.

(5) The stepper motor required a minimum of at least 2 milliseconds between each step. The stepper motor, whose rotations take at least 20 milliseconds, is the component of the system that moves the most slowly. Additionally, the ToF sensor requires the stepper motor to move at that speed in order to accurately measure distances, causing the system to move slowly as well.

## Circuit Schematic

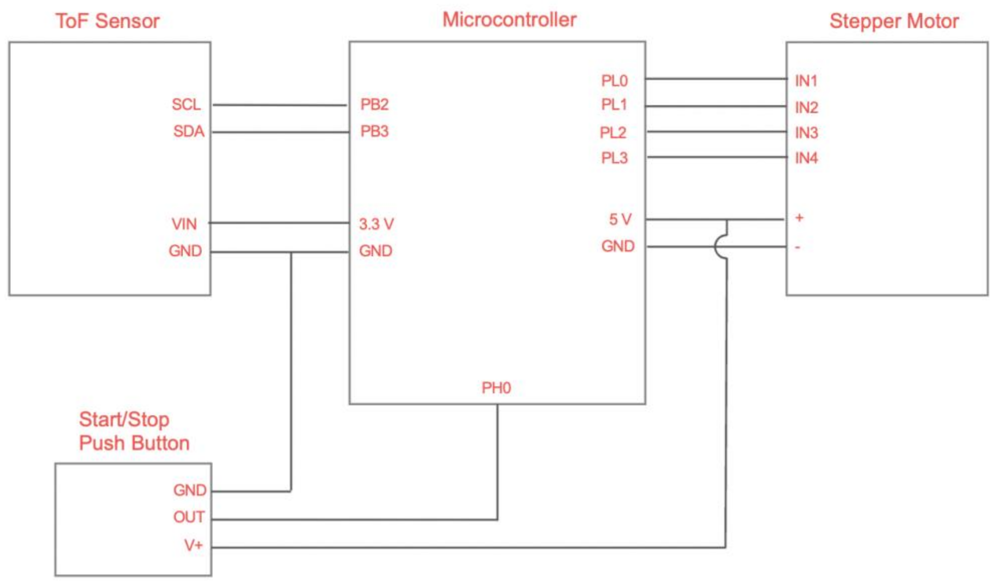

*Figure 5 – Circuit Schematic*

## Programming Logic Flowcharts

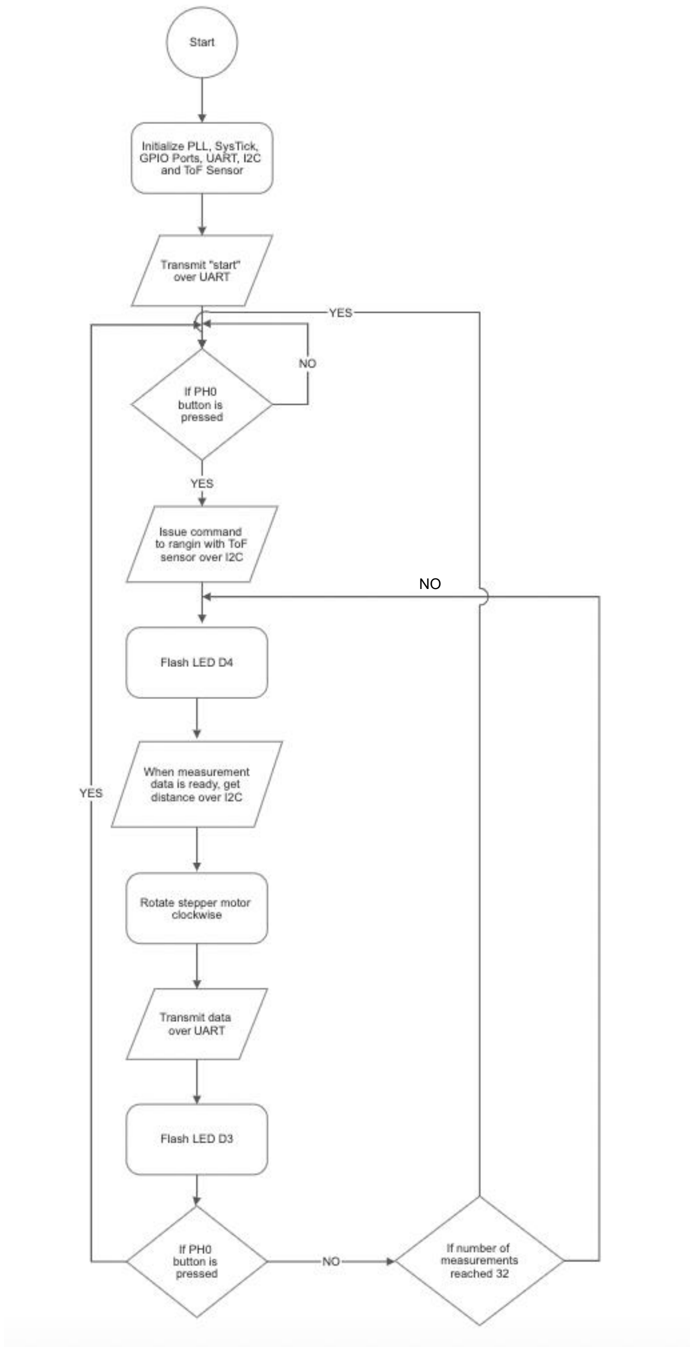
Start
Initialize PLL, SysTick, GPIO Ports, UART, I2C and ToF Sensor
Transmit "start" over UART
YES
NO
If PH0 button is pressed
YES
Issue command to rangin with ToF sensor over I2C
NO
Flash LED D4
When measurement data is ready, get distance over I2C
YES
Rotate stepper motor clockwise
Transmit data over UART
Flash LED D3
If PH0 button is pressed
NO
If number of measurements reached 32

*Figure 6 – Flowchart of Data Collection*

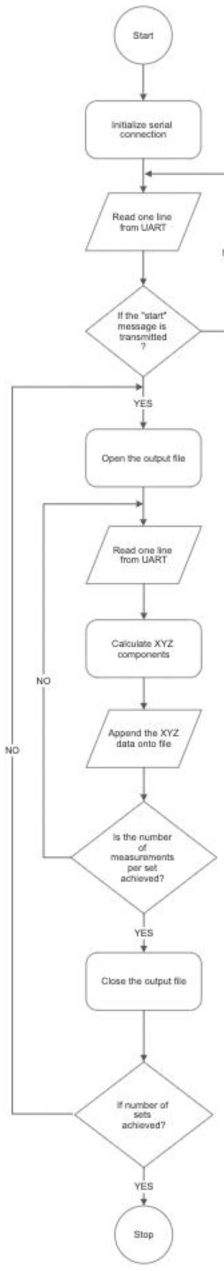
Start
Initialize serial connection
Read one line from UART
NO
If the "start" message is transmitted ?
YES
Open the output file
Read one line from UART
Calculate XYZ components
NO
NO
Append the XYZ data onto file
Is the number of measurements per set achieved?
YES
Close the output file
If number of sets achieved?
YES
Stop

*Figure 7 – Flowchart of PC Data Collection*

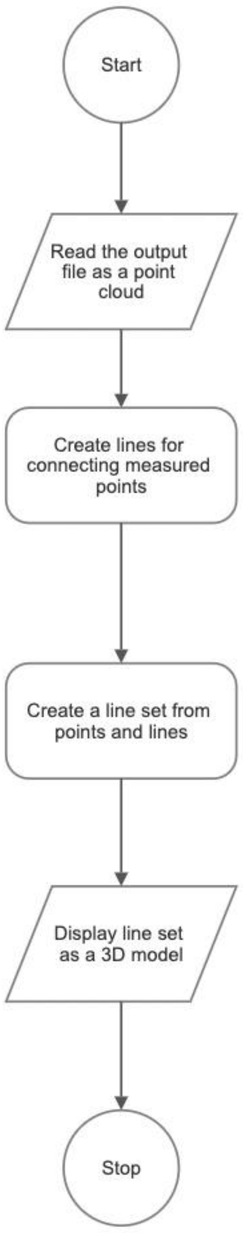
Start
Read the output file as a point cloud
Create lines for connecting measured points
Create a line set from points and lines
Display line set as a 3D model
Stop

*Figure 8 – Flowchart of PC Visualization*